A lifetime as an amateur compositor

David Walden

The first section of this paper briefly relates my experience writing and printing documents until I began to use TEX. The second section summarizes why I now use TEX and gives examples of its benefits, particularly writing books. Section 3 touches on the advantage of being able to use a separate powerful text editor, since TEX does not require use of a built in editor. Go straight to section 2 if you want skip my reminiscences that are not directly related to TEX.

1 First fifty years

1.1 Pre-computers

For some reason, I have always been interested in putting marks on paper — as with many people, my first work was with crayons, finger paints, and 1 inch, ruled "primary paper" and thick "primary pencils."[1] But it was not long before I moved on to more publication-like processes. Our church had a mimeograph machine and my parents were involved with producing the Sunday programs, and my parents both taught in the public schools[2] where they prepared handouts to students using ditto machines. I was a little involved with reproducing such materials at least throughout my teen years.

My father had an Underwood manual typewriter upon which he typed and on which I banged with a few fingers as a child. Later, he obtained a Royal manual typewriter on which I typed with ten fingers from the time I took a typing class two hours a week for one term during my sophomore year of high school. Ever since taking that typing class, I have been typing, often for reproduction, more or less constantly: so much so that when my son was a child and people asked him what his father did, my son's answer was, "He is a typist." After I went away to college, I moved to a Smith Corona electric portable typewriter; and when I entered the work world, we used IBM Selectric typewriters with their changeable type balls.

However, I wasn't a mistake-free typist, and I had much use for the tools of typewriter correction using carbon paper and other typewriter-base media: erasers, stuff to patch a mimeograph stencil, a razor blade to scrape the ink off of a ditto master, and KO-REC-TYPE paper and Snopake correction fluid to paint over typing so characters could be retyped correctly on pages that would be reproduced on Xerox copiers.

At `http://www.tpub-products.com/`, I found a document for sale that describes the duties of a military "religious program (RP) specialist" (an assistant or secretary to a chaplain), and it includes instructions for using Ditto masters; I quote it below. This description represents about the mid-level of complexity of pre-computer desktop "publishing" — more complicated than carbon paper (but not much) and slightly less complicated than a stencil machine.

> Before proceeding to an explanation of stencil preparation, the Ditto master will be discussed. The white Ditto master (overlay) is attached to a sheet of paper which is thickly coated with a carbon substance. Typing and hand- stylus impressions are made on the overlay and cause the carbon substance to be imprinted on the reverse side of the master. When the overlay is attached to the Ditto machine, the carbon-coated sheet is detached. The carbon impressions of the Ditto master are moistened by the duplicating fluid as the drum is rotated, which in turn transfers the carbon dye to the paper being fed into the machine. This transfer yields an exact reproduction of the master.
>
> Preparing a neat and accurate Ditto master stencil is one of the more important secretarial tasks that the RP will perform. Command Religious Program announcements are often distributed to command personnel through the use of Ditto copies. Just as the appearance of the office of the chaplain makes an instant and lasting impression, an information "flyer" or announcement will also leave lasting impressions. If the announcement is neatly prepared with concise and accurate information, it will probably give people the impression that the office of the chaplain is an efficient and caring organization. Therefore, it is important that the RP prepare each Ditto master with these thoughts in mind. The following helpful hints should aid the RP in preparing Ditto masters:
>
> • The "flimsy" sheet of paper that is inserted between the Ditto overlay and the carbon attachment MUST be removed before it is possible to

[1] The oral presentation version of this paper given at the PracTEX'06 conference included a number of photographs that are not included here because I did not seek permission to use them.

[2] Supported by the town government.

> have impressions transferred to the back of the overlay. NOTE: If there is some art-work involved, the "flimsy" may be left between the overlay and carbon attachment while the art-work is penciled lightly onto the overlay. The artwork can then be retraced with a stylus when the "flimsy" is removed. If an electric typewriter is being used, a test line should be typed on a Ditto master at each typing pressure setting. A copy should then be run and the RP can select the pressure that will provide the best copy. For manual typewriters, the typing pressure lever should be set to a medium or light position for best results.
>
> • A Ditto master should be left in the typewriter when errors are corrected. The typewriter platen should be turned until there is enough room to separate the perforated overlay from the carbon backing. A razor blade or other sharp-edged instrument should then be used to lightly scrape the carbon deposit of the incorrect characters from the back of the overlay. Next, a clean piece of Ditto carbon should be placed between the overlay and the original carbon. Then the typewriter should be returned to its original position and the correct letters typed. After the correction has been made, the temporary carbon that was used for this correction MUST be removed before proceeding.
>
> • Ditto masters may be reused at a later date if they are properly stored after the initial use. The masters should be placed in large envelopes and separated by flimsy sheets. It is imperative that they be stored in a flat position to keep them from becoming wrinkled

The point I am trying to make with the above discussion about the pre-computer era is that it took many (fussy, touchy, tedious) steps of careful work to produce good output, just as it does today in the world of fancy computer-based systems. Added problems were that the "desktop" (versus professional printing) approaches to reproducible typesetting in the pre-computer era didn't produce high quality printing, and there were limits on the number of copies you could make before the masters wore out.

1.2 Early computer use

I first came in contact with computers when I was in my junior year in college. While I still continued to use a typewriter for the next decade or so, I was also gradually changing over to using computers for typing documents, especially those that would be reproduced. I started with punch cards and an IBM 025 key punch machine, moved to rolls of punched paper tape with editing using Dan Murphy's TECO (tape editor and corrector), continued using TECO (modified to work with computer files rather than paper tape) via a Model 33 Teletype and then a TI Silent 700 as I moved into the world of computer time sharing (where the computer terminal was in my own office for the first time), used Jeremy Salter's RUNOFF (the first word processing program) on MIT's CTSS system, MRUNOFF (a version of RUNOFF for the TENEX operating system), and briefly touched troff/nroff in the early years of Unix. This computer-based world allow editing (e.g., with TECO) and reprinting of the actual raw text of a document or, eventually, inclusion of typesetting commands that would be interpreted by RUNOFF, MRUNOFF, and troff/nroff to produce the final document which could then be reproduced.

In the mid- to late-1970s, I first used a personal computer — an Apple II — but only to run VisiCalc. I was still doing word processing using MRUNOFF on TENEX. In 1981, IBM announced its PC and I got one for the following Christmas, I believe. My wife began using WordStar, and I helped her because I was familiar with command-based word processing from MRUNOFF which my friend Rob Barnaby (developer of WordStar) had also used.

1.3 Word and WYSIWYG

I don't remember when, but before too long (perhaps on the first PC AT) I began using the WYSIWYG PC-Word for DOS (based on the ideas of Charles Simonyi). Then I converted to using the Mac and MacWord which seemed to be where the forefront of Word development was taking place. MacWord was somewhat incompatible with PC-Word, but my PC-Word files converted over to the MacWord pretty well, although my memory is that the very straightforward style sheets of Word for DOS were no longer quite so straightforward with MacWord and I couldn't find some other features I had been using with PC-Word. I used MacWord for about eight years. In the early 1990s I decided to convert back to using an IBM PC using a Windows-based DOS operating system and then Win 3.1, but I discovered that my original Word files for the early PC that had been converted to MacWord did not convert back to the later versions of PC-Word very well. This was quite distressing to me. Moreover, after each of these changes I could not find various capabilities I was used to using — they were perhaps still there but apparently had moved or how they were executed had changed.

As time went by and I continued to use Word as part of Microsoft's Office Suite, I became increasingly annoyed at Word. Bigger, more complicated releases kept coming out, and in time there was pressure from people with whom I exchanged Word files to upgrade to the latest version because earlier versions couldn't

easily handle files from later versions without the person using the later version explicitly saving the files in the format of the earlier version, something many Word users didn't even know how to do. Also, each new release tended to again change how one called for various capabilities to be executed, while in time Microsoft stopped shipping hard copy manuals with Word from which one could learn such things (Microsoft increasingly forced users to depend on on-line documentation which doesn't work so well when you don't know how to ask for what you want to know about). Also, each new release tended to try to do more things automatically for me, and it took more and more work to turn off all the "help" it was trying to provide to me — help that in many cases actually made things harder for me (while it didn't help me by providing powerful editing functions, e.g., using regular expressions).

1.4 Breaking with Word; choosing TEX

My level of annoyance and frustration grew and grew, and eventually I made the decision to stop using Word for significant writing projects and to seek an alternative. Before I go on about my alternate approach, I must emphasize that I still use Word regularly for short, one-off projects (e.g., a short letter that I will not need to access on-line at a significantly later time) and when I work with someone who uses Word for his or her document preparation.

I chose to use LATEX as my alternative to using Word for document preparation for several reasons:

- It had a visible, non-proprietary, documented markup with a simple, plain text syntax that I was confident would allow me to reuse text in different documents over the years.[3]
- I was already familiar with command-based word processing and (as a computer programmer at heart) liked what I know about TEX's programmability. I also welcomed the prospect of being able to use a powerful text editor again as part of my document editing process (more about this in section 3).
- I had been involved with religious arguments about which of PageMaker, FrameMaker, or Interleaf was the best tool in various situations; and, from what I knew then, they also had some of the same problems as Word in terms of hidden markup and pressure on users to adopt new releases that obsoleted prior releases. I also was definitely looking for something that did not involve a graphical user interface (GUI) — something that required less mouse clicks. So, I didn't seriously investigate the just mentioned systems.
- I am a great admirer of Donald Knuth and thought it would be nice to try the system he developed.

Part of my preference for LATEX over Word comes from the fact that all of the markup is in a file where I can see it and change it rather than it having to be accessed by various menus and being largely unseen (except in its effect) in the document. To take a simple example, suppose I wanted to make the word "brown" in the phrase "quick brown fox" be bold. In Word I would select "brown" with the mouse, pull down the Format menu, click the Font item on the menu, click Bold in the Font Style column, click OK, and then the text would appear in the document in bold when displayed or printed (alternatively I could type control-B after selecting the word "brown"). To do the same thing in LATEX, I would change the text "quick brown fox" to "quick `\textbf{`brown`}` fox" with my text editor, and "brown" would display in bold when my LATEX file was compiled.

No doubt there are ways in Word to do many if not all of the things I now do with LATEX, but I find them mostly easier to find and do in LATEX.

As an aside, another aspect of Word that annoys me is that it is forever guessing what I want. For instance, if I type an explicit new-line (Return key), Word may decide to capitalize the first word of the next line, which may or may not be right. When I select some text with the mouse in Word, it often chooses different text than I touch with the mouse, for instance an extra space. Much or all of this can probably be turned off and I turn off as much as I can, but I never seem to be able to turn off everything; and, while Word's "help" sometimes does result in what I want, it seems more often to choose what I don't want. LATEX never seems to cause me this problem, which is not to say there are not other problems with LATEX.

I don't remember what TEX distribution — I downloaded something from the Internet — I tried first using NotePad on the PC for my editing. I do remember buying *The TEXbook*, and then quickly discovering LATEX which I experimented with a little bit. Then, I bought a copy of PCTEX on the theory that it would be nicely packaged, and I used it for a while but grew dissatisfied with the power of its editor. Then I found and downloaded WinEdt and MiKTEX.

[3] Word's hidden markup and WYSIWYG editing means that it is often hard to tell how something got to be the way it is. Also, since many Word users don't use style sheets, formatting (for instance, of a subsection title) might be done one way for one subsection and another way for another subsection, increasing the probability of inconsistencies in the output.

Later I bought and tried the Y&Y distribution, but I could never get it to work well; I did buy and make good use of the VTEX distribution for one particular project, but again I didn't like its editor. I ended up using WinEdt (and occasionally EMACS for things that seemed harder to do in WinEdt than in EMACS) and MiKTEX for a number of years, most recently obtained as part of TUG's ProTEXt distribution.

2 Why I use LATEX, particularly for writing books

Two reasons typically given for using LATEX are for its math support and for very nice looking typesetting. Neither of these is particularly important to me: I rarely have any math in my writing (but it is nice to be able to handle it easily in those rare cases where I do have it); I have a pretty undiscerning eye when it comes to typesetting, and what LATEX produces is more than good enough for me.

Here's what matters most to me about LATEX:

1. its programmability and modularity;
2. that I get to use a powerful editor with it;
3. that the mark-up is clearly visible to me and can be changed directly with a text editor;
4. its capabilities for explicitly specifying cross-references, maintaining bibliographies, and automatically numbering chapters, sections, figures, tables, footnotes, etc., which permit easy reorganization of text within documents and reuse in other documents
5. its relatively slow pace of change and great concern among the developers for backwards compatibility.

In other words, my use of LATEX is primarily about productivity. (Of course, there are certain limitations on this productivity such as when I finish writing a book using LATEX and the publisher tells me I must convert the text to Word and the figures to PowerPoint slides for input into the compositor's typesetting system.)

Much of my work using LATEX is on book length documents. For these I have compiled a more or less standard set of techniques that I feel help me be more efficient. I don't claim that the techniques I use are the techniques of a master; in fact, I view myself as an intermediate user of LATEX — I know enough to make LATEX jump through a few simple hoops, but not enough to know if my approaches are recommended or if they include some bad habits.

In my experience, publishers don't think much about the design of a book until they have the completed manuscript in hand. Since I use LATEX to develop the original manuscript, I have to make lots of temporary design decisions, and I want to be able to change these decisions with a minimum of work when the publisher does begin to deal with the design. Also, I am currently working on a book that I will be self-publishing, and settling on the design for this book is an iterative, experimental process where it is even more important to be able to make changes throughout the book (for instance, to the style of figure captions) with minimal work. My experience, however, should not prevent you from checking if the publisher of your document already has a standard style and perhaps even a LATEX class file that you can use from the outset of your writing. In any case, my emphasis here is *not* on the methods of representing preferences for appearance; my emphasis is on methods for easily and repeatedly *changing* the overall document appearance as well as on other methods for working efficiently on large documents.

Some of what I am about to describe for working efficiently on books or other long documents is probably already well known to many readers; perhaps you can make suggestions for how I might do things better.

(At several points in the following, I have included in parentheses discussions of basic TEX and LATEX issues that reviewers and others who have read drafts of this paper have asked me about that are not actually on the subject of book-writing productivity. Perhaps these parenthetical notes should have been footnotes, but I was too lazy to deal with the need for alternatives to `\verb` in footnotes.)

2.1 Include files

Suppose I am working on a book entitled *Breakthrough Management*, as I have been recently. I created a top level file named `bt.tex` with the following contents:

```
\documentclass{btbook}
\begin{document}
\include{titlepages}
\include{preface}
\include{surviving}   % a chapter
\include{rapid}       % another chapter
. . .                 % more chapters
\include{acknowledgements}
\include{bibliography}
\include{bio}
\include{index}
\end{document}
```

The text from included files appears to LATEX as if it were in the file `bt.tex` in place of the `\include` commands. In this way, I contain the text related to each chapter and other parts of the book in its own file. I let LATEX take care of numbering the chapters

and figures (or whatever) within chapters. If I later decided to change the order of chapters, I just change the order of the `\include` commands in the `bt.tex` files, and LaTeX automatically renumbers everything.

To work on one chapter at a time, my file `bt.tex` evolved to include many `\includeonly` commands, e.g.,

```
\documentclass{btbook}
%\includeonly{preface}
%\includeonly{surviving}
\includeonly{rapid}
%includeonly{surviving,rapid}
. . .
\begin{document}
\include{titlepages}
\include{preface}
\include{surviving}   % a chapter
\include{rapid}       % another chapter
. . .                 % more chapters
\include{acknowledgements}
\include{bibliography}
\include{bio}
\include{index}
\end{document}
```

In the above example, only the file `rapid.tex` gets compiled when I run LaTeX on the file `bt.tex`. In this 10-chapter book I had a couple of dozen `\includeonly` commands in the `bt.tex` file that I could comment in and out to work on each chapter individually and with various combinations of related chapters.

(Since the `\include` commands result in text being typeset, they must follow the `\begin{document}` command. The `\includeonly` commands must go in the preamble or else LaTeX complains.)

2.2 Custom class file

I have created a file `btbook.cls` which is my own personal class file for this particular book. This file is processed when LaTeX sees, at the beginning of the file `bt.tex`, the command `\documentclass{btbook}`. The first three lines of the file

```
\NeedsTeXFormat{LaTeX2e}[1994/12/01]
\ProvidesClass{btbook}[2006/01/21 BTbookclass]
\LoadClass{book}
```

define the class for this book to be named 'btbook' and to be an augmentation of the LaTeX book class.

The rest of the lines of the file are read and executed when LaTeX is run as if they were lines of text immediately following the `\documentclass` command in the `bt.tex` file.

(If your publisher already provides a LaTeX class file, you can still collect all of the sorts of things I describe below in their own file and `\input` that file in the preamble rather than just putting all these things directly in your preamble. I prefer not to have much in my preamble beyond the `\includeonly{...}` commands that I am constantly commenting in and out.)

2.3 Packages

Next in the class file comes the list of packages I use for writing this book.

```
   % Palatino is basic roman font
\RequirePackage{mathpazo}
   % Helvetica is sans serif font
\RequirePackage[scaled=.95]{helvet}
   % Courier is typewriter font
\RequirePackage{courier}
   % for including images
\RequirePackage{graphicx}
   % for formatting URLs
\RequirePackage{url}
   %to be able to rotate figures
\RequirePackage[figuresright]{rotating}
   % for dropped caps
\RequirePackage{lettrine}
   % for tighter list spacing
\RequirePackage{paralist}
        \setlength{\pltopsep}{.05in}
   % for comment environment
\RequirePackage{comment}
   % for endnotes with reformatted numbers
\RequirePackage{dw-endnotes}
\RequirePackage{setspace} %\doublespacing
```

When I find I need to use another package, I add another `\RequirePackage` line to this list. (As I understand it, `RequirePackage` does the same job as `\usepackage` except it doesn't allow the same package to be loaded twice which apparently might cause problems in some cases.)

Notice that the package name in one case includes the characters `dw-`. This is my convention for noting a package that I have modified. In such cases, the file of the modified package is in the same directory with the rest of the files for this book or in the local changes part of my `texmf` data structure. I seldom understand a package I am modifying; I typically use a hit and miss approach to change stuff until I get the results I want.

Copy editors who edit on hard copy like double spacing, and I can provide that with a one character change — uncommenting the `\doublespacing` command on the last line above that loads the setspace package.

2.4 Miscellaneous useful macros

The following macros provide a few capabilities I use relatively frequently.

```
   % space around em-dashes
\newcommand{\Dash}{\thinspace---\thinspace}
   % mark text that needs checking
\newcommand{\CK}[1]{\textbf{CK #1}}
   % marginal note to myself
\newcommand{\manote}[1]{%
  \marginpar{\scriptsize To do:\\#1}}
   % cut-in title
\newcommand{\partitle}[1]{%
  \medskip\noindent\textbf{#1}}
   %change function of \url command
\let\Originalurl=\url
\def\url#1{{\fontsize{10.7pt}{13.05pt}
  \Originalurl{#1}}}
```

For some documents I have worked on, I have had many more such miscellaneous useful macros.

Anyone trying to improve productivity using LaTeX who doesn't already define his or her own macros should learn to do so. User-defined macros allow significant improvements in efficiency. For instance, the first macro above defines the command `\Dash{}` to be an abbreviation for the character string `\thinspace---\thinspace` which results in an em-dash being typeset with a *little* bit of space on each side of it, as in aaa — bbb. It is less characters and probably more reliable to type `\Dash{}` many times in a book than it is to type the characters `\thinspace---\thinspace{}` many times. In my view, however, the greater benefit of defining the `\Dash` command comes when my publisher tells me that its style is closed-form em-dashes (no space on either side, i.e., aaa—bbb) or a more open form (aaa — bbb). To implement either of these changes *throughout* the book, I merely redefine `\Dash`, e.g.,

```
   % no spaces around em-dashes
\newcommand{\Dash}{---}
```

or

```
   % full spaces around em-dashes
\newcommand{\Dash}{ --- }
```

and recompile my document. Containing such style conventions within a few lines of a large document and being able to change the style throughout the document with only a few key strokes is an enormous advantage. (I'll give a more complex example of such containment when I discuss macros for figures and tables below.)

To redefine a command that already exists in LaTeX or has been defined by a package that has already been loaded, for instance to define a variation on `\url` as I do in the last two lines of my group of miscellaneous useful macros, I have to use the `\renewcommand` command. The `\renewcommand` works just like `\newcommand` except that LaTeX does not complain with `\renewcommand` if I try to give a definition to a command that already exists — a good thing to be warned about when one uses `\newcommand`. (In the next subsection I give another redefinition example — redefining `\footnote`.)

2.5 Footnotes and endnotes

In the case of `\RequirePackage{dw-endnotes}`, I am using the `endnotes` package, modified slightly to change the format of the note numbers.

Typically, I put footnotes on the bottom of text pages where they are referenced, at least while I am drafting chapters and want to be able to see the notes without having to turn a bunch of pages. However, publishers tend not to like having footnotes — it makes a book look too academic to be popular, in their view. Thus, before actual publication, I often find myself converting all my footnotes to endnotes. The next commands in my class file do this.

```
   %comment out to not have end notes
\renewcommand{\footnote}{\endnote}

\newcommand{\dumpendnotes}{%
  \medskip
  \begingroup
    \setlength{\parindent}{0pt}%
    \setlength{\parskip}{1ex}%
    \renewcommand{\enotesize}{\normalsize}%
    \theendnotes
  \endgroup
  \setcounter{endnote}{0}}
```

First, the `\footnote` command is redefined to be the `\endnote` command; this avoids my having to replace every instance of `\footnote` with `\endnote`. Then the class file defines a command (`\dumpendnotes`) that can go at the end of each chapter to dump the chapter's endnotes, formatted as I want them to be. If the command `\dumpendnotes` was already defined in LaTeX or some other package, LaTeX would warn me because I didn't do the definition with `\renewcommand`.

2.6 Formatting figures and tables

The next set of commands in the class file have to do with changing the format of figure and table captions without actually modifying a LaTeX or package file. The LaTeX default does not use bold face for captions and uses a period rather than a hyphen between the chapter number and figure number within a chapter. The following changes patch LaTeX to follow my preference for bold face and hyphens.

```
\long\def\@makecaption#1#2{%
  \vskip\abovecaptionskip
  \sbox\@tempboxa{\textbf{#1}. \textbf{#2}}%
  \ifdim \wd\@tempboxa >\hsize
    {\textbf{#1}. \textbf{#2}\par}
```

```
  \else
    \global \@minipagefalse
    \hb@xt@\hsize{\hfil\box\@tempboxa\hfil}%
  \fi
  \vskip\belowcaptionskip}
\renewcommand \thefigure
 {\ifnum \c@chapter>\z@ \mbox{\thechapter-}%
 \fi\@arabic\c@figure}
\renewcommand \thetable
 {\ifnum \c@chapter>\z@ \mbox{\thechapter-}%
 \fi\@arabic\c@table}
```

(I do not have to bracket these lines, top and bottom, with `\makeatletter` and `makeatother` commands as I would have to if this patch was in the preamble of my document; the at-sign is a letter by default in class files and packages. Some readers may be back a step, at the question of, "What is it about at-signs anyway?". The answer is that the files for basic LaTeX, for class files, and for other packages are full of macros names that include an at-sign (@), e.g., a macro named `\@makecaption` is defined at the beginning of the above example. My understanding is that an at-sign is used in low level programming of LaTeX, class files, and packages to create macro names that can't accidentally conflict with names that may be defined by users not doing such LaTeX "systems programming." An at-sign is normally not a letter and thus cannot be part of a macro name. However, in the above example I want to patch low level LaTeX code that includes at-signs in its macro names; if I was trying to make this patch in my preamble (as I used to do before I learned to make some patches in a personal class file), I would have to tell LaTeX to temporarily turn at-signs into letters, make the patch, and then turn them back into non letters (other) so the rest of my program could use @ in the normal way where it is not a special character of any kind.)

Perhaps there is a caption package that would allow such changes without patching LaTeX, but I was shown how to make this patch a few years ago and it works, so why bother trying to find and learn a new package?

Next in my class file comes a set of definitions for commands I use to include graphics. I seldom insert `\begin{figure}` and `\end{figure}` commands directly into my documents; I do, from time to time, insert the commands `\begin{table}` and `\end{table}`. It is inevitable that, before I am done with a big document, I will want to change the formatting relating all figure and tables — perhaps several times. Thus, I use macros for inserting almost all figures (or tables) such that I can make changes to formatting relating to the figures by making changes to only a few lines in the relevant macros.

```
   %switch argument among pdf, eps, etc.
\newcommand{\figfiletype}{pdf}
   %tell LaTeX directory path to figures
\graphicspath{{figures/}}
   %commands to display file name, or not
\newcommand{\DFN}[2]{%
  \texttt{\small[#1 #2]}}
%\newcommand{\DFN}[2]{}

\newcommand{\snfig}[3]{%scaled numbered figure
   %drop htb and %s for single page figures
   \begin{figure}[htbp]
%\vbox to \vsize{%
    \hfil\scalebox{#3}{
     \includegraphics{#2.\figfiletype}}\hfil
   \caption{\label{fig:#2}#1 \DFN{#2}{#3}}
%\vfil
%}
   \end{figure}
   }

\newcommand{\sntab}[3]{%scaled numbered tables
   \begin{table}[thbp]
%\vbox to \vsize{%
   \centering
   \caption{\label{tab:#2}#1 \DFN{#2}{#3}}
   \smallskip
   \scalebox{#3}{
     \includegraphics{#2.\figfiletype}}
   \label{tab:#2}
%\vfil
%
   \end{table}
   }

\newcommand{\unfig}[2]{%scaled unnumbered fig.
   \begin{figure}[htbp]
   \hfil\scalebox{#2}{
     \includegraphics{#1.\figfiletype}}\hfil
   \label{fig:#1}\centerline{
     \DFN{#1}{#2}}
   \end{figure}
   }

   %sideways scaled numbered figure
\newcommand{\swsnfig}[3]{
   \begin{sidewaysfigure}
    \centering
    \scalebox{#3}{
     \includegraphics{#2.\figfiletype}}
   \caption{\label{fig:#2}#1 \DFN{#2}{#3}}
   \end{sidewaysfigure}
   }
```

For instance, the macro `\snfig` above takes three arguments. The text for a figure caption, the

unique part of the file name for the graphic to be included, and a scale factor for the graphic, e.g.,[4]

```
\snfig{This is the caption}{figure3-31}{.8}
```

The full name of the file to be included is the concatenation of the part of the file name that came from the second argument of the macro call, the directory that is specified by the `\graphicspath` command (an option of the graphicx package) as the place LaTeX searches for figures, and the `\figfiletype` definition as the file name extension. The latter is useful because sometimes all of my figures are `.eps` files and sometimes they are `.pdf` files, and sometimes I switch between these two formats at different times in the production of the book. (When using `.eps` format, I compile using LaTeX and a dvi-to-pdf conversion; when using `.pdf` format, I use pdfTeX to compile. If the graphic format was changing from file to file within the document, I would instead specify the format as another argument to the `\snfig` command. [However, Will Robertson and others have recently pointed out to me that if I leave the extension off, `\includegraphics` will pick the appropriate extension: `.eps` for LaTeX and `.pdf` for pdfLaTeX.])

While I am drafting and revising a book manuscript, I want to be able to look at a figure in the printed output and know what file I need to modify to change the figure. Thus, my macros for including figures and tables causes the file name to be included in the printed output in small letters enclosed in small square brackets, using the macro `\DFN`. When it comes time to create the final manuscript, I swap to a definition of `\DFN` that produces nothing and recompile the book's LaTeX files.

The definitions of `\snfig` and `\sntab` also include several lines that are commented out. Professional editors often like to see the manuscript with figures or tables each on its own page rather than in-line with the text. Commenting in these few lines puts the figures and tables of the whole book on their own pages.

The `\snfig`, `\sntab`, and `\swsnfig` macros also define labels for cross-referencing the figures with `\ref` or `\pageref` commands. A slight limitation of my implementation is that I cannot reuse the same figure or table file without confusing the labeling. However, it is easy enough to create a duplicate figure or table with a different file name.

I typically create all figures and most tables outside of TeX itself and include them from separate files. If I found myself inserting very many tables directly into my `.tex` files rather than including them from graphics files, I would define a `mytable` environment so that I could still contain and simply change the sort of formatting I have discussed.

2.7 Thought breaks

The next group of commands (mostly commented out) are various options for indicating what I call "thought breaks" — places where formatting indicates a change of topic big enough to highlight but not big enough to have its own section or subsection title.[5] (These commands are defined with `\def` because I know they will pick up the correct arguments this way, and I am not sure enough of the details of how `\newcommand` works. I understand the details of how TeX defines a macro and then collects its arguments when the macros are called because Knuth explains it pretty completely in *The TeXbook*. In particular, TeX allows macro calls where the arguments of the macro are not all embedded in pairs of braces. However, I have never stumbled across a rigorous explanation of how a macro defined with `\newcommand` collects its arguments and thus in what situations arguments not in braces will be recognized or to what extent LaTeX defined macros can have both of what Knuth calls delimited and undelimited arguments — and I have not bothered to study the LaTeX code to figure it out. Consequently, out of ignorance, I use `\def` to define macros which don't have their arguments delimited by braces.)

```
\begin{comment}
\def\newthoughtgroup#1{\bgroup
 \afterassignment\BigFirstLetter \let\next=}
\def\BigFirstLetter#1{
 \bigskip\noindex{\Large#1}}

  %adapted slightly from Victor Eijkhout on ctt
\def\newthoughtgroup#1{\BigFirstLetter#1$}
\def\BigFirstLetter#1#2$${
 \bigskip\noindent{\Large #1}#2}
\end{comment}

\def\newthoughtgroup#1{%
   \bigskip\noindent {\large #1}}

\begin{comment}
\def\newthoughtgroup{%
```

[4] Barbara Beeton pointed out to me at the PracTeX'06 conference that `\includegraphics` can take a scale factor as an optional argument, and thus I don't need a separate `\scalebox` in the above definitions. I believe the `\scalebox` commands remain from before I switched from using the graphics package to using the graphicx package.

Also, I began using the `\hfil` commands in the above definitions before I understood I could use `\centering` as in the last definition.

[5] See my *PracTeX Journal* 2005-4 "Travels in TeX Land" column (`http://www.tug.org/pracjourn/2005-4/walden/`) for examples of thought breaks.

```
  \bigskip\noindent }

 %big bold dropped cap letter with rest
 %  of word small caps
\def\newthoughtgroup#1#2 {
 \bigskip\noindent\lettrine{#1}{#2}\ }

\def\thoughtbreak{\vskip2pt
  \centerline{$^{\vrule width2cm height 1pt}$}
   \vskip2pt\noindent}
\end{comment}
```

The version of `\mythoughtgroup` currently not commented out indicates the new thought by a vertical space and a slightly bigger capital letter at the beginning of a non-indented paragraph.

2.8 Chapter formatting

The final set of commands in my class file has to do with the beginnings and ends of chapters. At the beginning and ending of each chapter I insert some commands that I can change either by changing the commands themselves or changing macros in the class file.[6]

```
\RequirePackage{fancyhdr}\pagestyle{fancyplain}
\newcommand{\mypartname}{}
\newcommand{\mychaptername}{}
\lhead[\fancyplain{}{
 \thepage}]{\fancyplain{}{}}
\chead[\fancyplain{}{
 \mypartname}]{\fancyplain{}{\mychaptername}}
\cfoot[\fancyplain{}{}]{
 \fancyplain{\thepage}{}}
\rhead[\fancyplain{}{}]{
 \fancyplain{}{\thepage}}
\newcommand{\EMPTYPAGE}{\clearpage
 \thispagestyle{empty}\cleardoublepage}

\newcommand{\ENDCHAPTER}{\dumpendnotes}
\newcommand{\fENDCHAPTER}{\vfil\dumpendnotes}
```

In the class file for the book from which I drew these illustrations, there are a couple of alternative macros that I can include at the end of each chapter to dump the endnotes, but the end-of-chapter macros could be defined to cause other actions and outputs. In this book (which has only 10 chapters) I do not combine everything in a single beginning-of-chapter macro (e.g., `\BEGINCHAPTER`), but I have done this with some books (e.g., the 20-chapter book I am also currently working on). The typical beginning-of-chapter commands for the chapter with the file name `rapid.tex` (mentioned earlier) are

```
\EMPTYPAGE
\chapter{Rapid Change in a Global World}
\label{ch:rapid}
\renewcommand{\mychaptername}{Chapter %
 \thechapter: Rapid Change in a Global World}
```

2.9 Using a fully developed class

For some books I have included different or additional capabilities in my custom class file.

Obviously, I could also use a fully developed class such as memoir rather than making lots of modifications of my own to I would still use some of the ideas I have described above. the LaTeX's standard book class. However, I suspect

It is clear that TeX and its derivatives with their explicit, visible markup provide a strong base for incrementally building a personal library of techniques that are easy to apply from one project to the next.

3 Possible benefits of a separate editor

Using a word processor such as MS Word that has invisible, undocumented, proprietary markup means you have to use its built-in, WYSIWYG editor that knows about that markup. This has two potential disadvantages: (1) GUI-based editing often takes a lot more key strokes to do simple things than an editor like WinEdt or EMACS (I gave an example of this in section 1 and provide additional examples below); (2) an editor like MS Word's does not seem to have a lot of useful features that an editor like WinEdt or EMACS has.

Many of the ideas in this section are probably relevant to any typesetting system that has editing capability (like those I describe below). For all I know, MS Word can do many of these things; but, as I said in the first section, a number of years ago I lost interest in struggling to find stuff buried in all Word's menus, dealing with its ever changing user interface, and its planned-obsolescence-and-forced-upgrades business strategy.[7]

3.1 Two ways to make a change throughout a document

In the last section I sketched the benefits of using macros for some sequences of commands (for instance, `\Dash{}` for `---`) that enable the replacement sequence to be changed everywhere in a document by

[6] When I first started customizing my page headings a few years ago, I used the fancyheadings package; recently I learned that the package fancyhdr has replaced fancyheadings, but I have not yet bothered to rewrite all the heading commands to use the new forms that come with the fancyhdr package and don't use the fancyplain device.

[7] The content of this section also appeared in a slightly different version in issue 2006-4 of *The PracTeX Journal* (`http://www.tug.org/pracjourn/2006-4/walden/`. That paper had additional content by Yuri Robbers on a number of the editors that work well with TeX.

just changing the definition of the macro in one place. Another option for making a change to the same sequence of characters throughout a document is to use a text editor's Replace All command. For instance, suppose I hadn't used a macro for em-dashes and instead had closed form instances of `---` throughout my document, e.g., "this is the end—the end of the line." And then suppose I decide to change the style to uses semi-open form em-dashes, e.g., "this is the end — the end of the line." With my editor I can do a Replace All of `---` by `\thinspace---\thinspace{}`. If the document is broken up into separate files for each chapter, it will be good if the text editor has the option for doing the Replace All over all documents open in the editor instead of only in the document where the cursor currently is.

Here is another example of a simple text replacement of the entire document. Suppose I decide (for some reason) to replace all en-dashes by hyphens. Then I can do the following sequence of three steps (the first and third steps are to avoid accidentally changing instances of `---` into `--`):

Replace All `---` with `#X#X#`
Replace All `--` with `-`
Replace All `#X#X#` with `---`

Now suppose I want to add a fourth argument to every instance of the macro call `\snfig{ }{ }{ }` (see definition and discussion of this macro in subsection 2.6), that is, change the macro call formats to `\snfig{ }{ }{ }{ }`. Of course, one approach is to search for each instance of `\snfig{`, then move the cursor to after the third pair of braces, and then type the fourth pair of braces. However, if your text editor has a capability for dealing with regular expressions, you can make this change more easily (the last book I wrote had a couple of hundred instances of `\snfig`).

While I won't get into the specific format of any particular editor's representation of regular expressions, they would do something like the following in our example case.

Replace All `\snfig{`*`(.*)`*`}{`*`(.*)`*`}{`*`(.*)`*`}`
with `\snfig{`***$1***`}{`***$2***`}{`***$3***`}{}`

Everything in the Replace All command that is in a typewriter format is literal characters to be replaced. The characters in italics in the first part of the command are special characters that match any number of characters between balanced braces. For instance, in the command

`\snfig{Caption title.}{file-name}{.5}`

the first instance of *(.*)* would match the first argument `Caption title`. The second instance of *(.*)* would match `file-name`, and the third instance would match `.5`. Better than that, the editor stores each matched set of characters in its own place for later reuse, as in the second half of the above command. The part of the command after the word "with" says to replace the `\snfig` command that was matched with all the same literal characters for the command names and braces, but to put the first match text in place of ***$1***, the second match text in place of ***$2***, and the third matched text in place of ***$3***, and to add an extra pair of literal braces at the end of the replacement. Thus,

`\snfig{Caption title.}{file-name}{.5}`

is turned into

`\snfig{Caption title.}{file-name}{.5}{}`

and the same is done for every other instance of `\snfig`, in each case properly maintaining the argument text through the replacement step.

The above example may need some tweaking if the instances of the command being changed sometimes span line boundaries, but typically this also can be handled, as can be much more complex instances of detecting what should be replaced and what should not be in various instances. In fact, depending on the editor's particular regular expression capability, the earlier example of replacing en-dashes by hyphens perhaps could have been done with one Replace All using a regular expression to search of `--` *not* followed by a third `-`.

In section two I recommended that anyone not already using LaTeX macros should learn to use them. I recommend the same thing about using regular expressions if your editor supports them. They won't be needed as often as macros, but when they are needed they are a major productivity increaser.

3.2 Other editor features

All of the serious text editors I have used allowed me to mark the cursor position with a couple of key strokes (e.g., Alt-F11 in WinEdt), move the cursor somewhere else (for instance to select some distant text and cut it, and then jump back to the first cursor position (e.g., Cntl-F11), where I might paste the text cut from elsewhere in the document.[8]

Text editors such as I have in mind also typically have provision to have multiple text buffers rather

[8] I don't claim that none of these features are available in various editors that are packaged with commercial versions of TeX; I hope they are. I am only suggesting that you find an editor that supports such capabilities. What I do know is that with each new release of MS Word I find it harder and harder to find such features, if they exist at all, while they are easy to find in the two text editors I currently use regularly (WinEdt and EMACS).

than just one cut-and-paste clipboard.[9] I gave an example of this in the regular expression example in subsection 3.1, where three bits of text were simultaneously saved from the replaced string of characters for placement in the replacement string. With multiple places to save text, it is also possible, for example, to search-for-and-cut one bit of text, search-for-and-cut another bit of text, search-for-and-cut a third bit of text, and then paste together at some other point in the file, saving (in this example) several moves of the cursor in comparison with an editor with a single clipboard.

To give another example of the usefulness of multiple text buffers, I often find something on a web page and want to copy something from the page as a quote in a document I am writing *and* copy the URL as the source of the quote. Without multiple text buffers this requires the following sequence: (1) select text to be copied, (2) copy to clipboard, (3) switch window to other document, (4) position cursor, paste contents of clipboard, (5) switch window for first document, (6) select URL text, (7) copy to clipboard, (8) switch window to other document, (9) position cursor, (10) paste contents of clipboard.[10] With multiple text buffers it requires: (1) select text to be copied, (2) copy to buffer A, (3) select URL text, (4) copy to buffer B, (4) switch window to other document, (5) position cursor, (6) paste contents of buffer A, (7) reposition cursor, (8) paste contents of buffer B. The latter method is not necessarily less key strokes (the macros I have for WinEdt take 12 steps), but it is somehow easier for me not to switch windows and have to refind my place as often.

The fact that LaTeX is not locked to a particular editor also means that each participant in a collaborative project can use the editor with which he or she is most familiar. (Collaboration is also made easier because there is much more compatibility from release to release of TeX and LaTeX, even with multiple providers, than there is from release to release with many non-TeX commercial products. For instance, MS Word seems to go out of its way to enforce inter-release incompatibility in a way apparently aimed at forcing all collaborators to all upgrade to the same release.)

Using a text editor in conjunction with LaTeX with its explicit markup also has advantages. For instance, it is easy to search for an italicized version of a word (i.e., search for `\textit{word}`) as distinct from a non-italicized version of the same word. Similarly, it is possible to search for all headings of a certain level (for instance, all instances of `\subsection`); with a system with implicit markup (e.g., MS Word) one might have to search for the words of each subsection title.

In subsection 2.1 I showed the use of `\include` files. This works because LaTeX has the provision for specifying in one file a list of files to be included as if they were text in that first file. The editor I mostly use, WinEdt, also supports this; it knows enough about LaTeX to search the highest level file for instances of `\include` and gives me a list of visual tabs to the various files to be included; this makes it very easy for me to move among the various files in a longer document.

I could give an unlimited number of examples of what powerful text editors can do once one breaks free of the limitations of hidden, proprietary, undocumented markup and those built-in editors whose graphical user interfaces eliminate powerful editing capabilities as part of providing a point-and-click environment to the user. One of the best things is that I can use the same editor with which I have become facile from application to application. (Also, if several authors are collaborating, they can all use their own preferred editor, and — perhaps more importantly — they don't all have to have the same version of Word installed.)

4 Conclusion

To conclude, I give some additional opinions on a couple of thoughts I hinted at in the earlier sections and one additional opinion.[11]

4.1 Conservation of hassle

In my observations covering many decades, it has always taken many fussy steps to do anything involving typesetting for reproduction. The computer era has eliminated many physical steps, each of which required its own sort of skill and complexity. However, the computer era hasn't done anything to decrease the total number of steps — they are just done with a keyboard and mouse now and the skill is in knowing what commands can do what you need and where to find them. What computer-based approach is best is a matter of personal choice — they are all filled with hassle. My own choice has evolved to be a powerful, explicit typesetting language (LaTeX) combined with

[9] Alex Simonic, the developer of WinEdt, the editor I mostly use, showed me how to write macros to provide multiple text buffers in WinEdt.

[10] I know I could have two windows open at the same time, but sizing the windows so both can be seen takes too many steps unless I am going to do a lot of copying between two documents.

[11] For a related point of view to some of what I say here, see "LaTeX for Windows — A User's Perspective", Proceedings of the 2001 Annual Meeting, *TUGboat*, Volume 22(2001), No. 3, pp. 140–145.

a powerful text editor. Some people argue that the WYSIWYG approach is easier. On average, however, I see the WYSIWYG approach as just as much work for the same task: A simple task in Word also tends to be simple in LaTeX; a more advanced task (e.g., inserting a cross-reference) seems more straightforward to do in LaTeX than in Word. As one uses more and more of LaTeX's power, the same task typically seems more and more difficult to do in Word as well, and my frustration level seems to grow faster with Word. Since most users use Word in only trivial ways, Word is pretty trivial to use at that level, but then so is LaTeX at that level.

4.2 The assertion that LaTeX is hard to learn

I have no doubt that most people could learn to use LaTeX and a good text editor if they saw it as beneficial; LaTeX and a text editor at the intermediate level of sophistication at which I use them are no more complex than trying to use Word for the same task (and I mostly think they are less complex than Word). Think about all the other, fairly complex things people master in their lives — cooking, knitting, growing flowers, fly tying, and the rules of baseball. By comparison, there is nothing inherently too difficult about using LaTeX — it's only a question of learning enough to see its comparative benefits (and cheaper price). And anyone who can learn to use Word at a high level with all its particular weirdnesses (and changes to the user interface with each release) can surely also learn to use LaTeX at the same level.

4.3 Using what everyone else is using

I believe the main argument against LaTeX and for Word is ubiquity of use. People use Word because everyone else does — their collaborators, their publishers, etc. — not because Word is better. If people were interested in a better word processor, they would use Word Perfect or perhaps one specialized to their area of writing such as Note Bene.

If the world of TeX is to have the best chance of snagging and keeping potential LaTeX users and users of other TeX-related system, we must continue to offer them a great, if small, community of fellow users. This is what volunteer-supported capabilities such as the TeX FAQ, CTAN, discussion groups such as `comp.text.tex` and `texhax`, TUG and the other volunteer-based user groups, the volunteer-created hardcopy and on-line journals, and meetings such as this PracTeX conference are about. It is a great pleasure for me to participate in this welcoming and informative community. Thank you.

Acknowledgements

I owe thanks to many sources for what I have learned about using LaTeX — books, the `comp.text.tex` list, the `texhax` list, and many individuals. Of course, none of them are responsible for lessons I have mislearned.

I can't remember and acknowledge everyone who directed me to techniques illustrated in this column; however, I can remember some of them. Karl Berry reviewed an early version of this paper and earlier told me about some of the methods I have described here. I also remember Peter Flynn, Steve Peter, and Steve Schwartz telling me about particular techniques. Peter Flom, Will Robertson, and anonymous reviewers provided many helpful suggestions for sections 2 and which appeared in earlier incarnations in issues 2006-2 and 2006-4 of *The PracTeX Journal*. Will Robertson also carefully reviewed the complete paper here and made many substantive suggestions for improvement as well as catching many minor errors.

Biographical note

David Walden is retired after a career as an engineer, engineering manager, and general manager involved with research and development of computer and other high tech systems. More history is at `www.walden-family.com/dave`.